Privacy-Sensitive Parallel Split Learning

Joohyung Jeon[1] and Joongheon Kim[2]
[1]*School of Computer Science and Engineering, Chung-Ang University, Seoul, Korea*
[2]*Artificial Intelligence and Mobility Lab., School of Electrical Engineering, Korea University, Seoul, Korea*
[2]joongheon@korea.ac.kr

***Abstract*—Mobile devices and medical centers have access to rich data that is suitable for training deep learning models. However, these highly distributed datasets are privacy sensitive making privacy issues for applying deep learning techniques to the problem at hand. Split Learning can solve these data privacy problems, but the possibility of overfitting exists because each node doesn't train in parallel but in a sequential manner. In this paper, we propose a parallel split learning method that prevents overfitting due to differences in a training order and data size by the node. Our method selects mini-batch size considering the amount of local data on each node and synchronizes the layers that nodes have during the training process so that all nodes can use the equivalent deep learning model when the training is complete.**

***Index Terms*—Distributed Deep Learning, Split Learning, Federated Learning**

I. Introduction

The traditional distributed deep learning algorithms [1], [2] train the deep learning model by using the data gathered in the data center to distribute the computation to the workers. However, data gathered in the mobile device, medical platform, and financial platform that cannot be shared or transported externally due to privacy issues must be used in model training in a different method [3]. Although methods such as Federated Learning [4] and Split Learning can be used to train models while keeping data privacy, the federated learning technique can be difficult to use when the model size is very large or when there is less computing resource on edge node. Split learning has very low computing requirements and communication requirements because only the least number of layers in the entire model that can protect the security of the data needs to be computed in the edge node [5]. However, there is a concern of overfitting due to differences in the training sequence and the local data size of the node, since the existing split learning method is conducted in the order of each node.

In this paper, the selection of mini-batch size by node considering the local data size of each edge node and the parallel split learning method considering the total distributed data are proposed. It also ensures that all edge nodes have the same model after training ends through the synchronization of the hidden layer that edge nodes have.

II. Proposed Parallel Method

A. Related Work

To preserve privacy in model training, split learning [6], [7], [8] splits the layers of the entire model. Each edge node(client) has the first hidden layer and its own local dataset. The central server has the remaining layers which is from second layer to output layer. Then the training process starts. We could denote each layer of client *k* as $L_{k,1}$ and layers in central server as $L_2, ..., L_m$. The first client and the central server randomly initializes the weight of $L_{1,1}$ and $L_2, ..., L_m$ respectively. The first client conducts standard forward propagation with its local data and transmit the output of hidden layer $L_{1,1}$ and label to the central server. Central Server propagates remaining layers with received output from the client. The central server generates gradients for its final layer and backpropagates the error until L_2. After that central server transmits the result of L_2 to the first client. The first client finally backpropagates the gradients in $L_{1,1}$ with received gradient from central server. After the first client finishes to use its local data for training the next client receives the layer weights from the first client and starts the training process with its local data. If training process of client *k-1* is finished, client *k* receives the weights of first layer of the client *k-1* and initializes its first layer weights with received weights. Then client *k* proceeds the training with its own data.

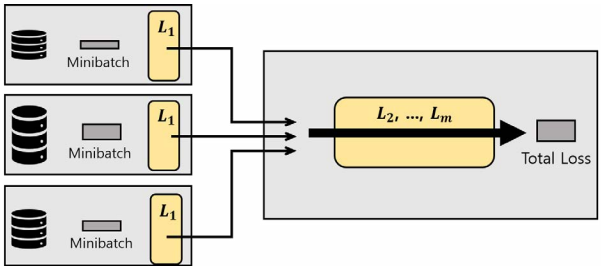

Fig. 1. System model where each client propagates with predefined mini-batch size which is proportional to its local data set.

B. Proposed Parallel Split Learning

Conventional Split Learning preserves privacy of client local data and requires low computational, communication resources than traditional distributed deep learning methods. However as the training process through each client is sequential, overfitting problem could occur. To address the issue of sequential split learning our method train the local data of each client in parallel.

Minibatch Size Selection. To use the local data of each client in parallel, our method set the mini-batch size of each client proportional to its local data size. We assume client *k* has local dataset $\mathcal{P}_k$, with $n_k = |\mathcal{P}_k|$. Then mini-batch size for each client *k* is b_k which is proportional to n_k. As shown in Fig. 1 in

each iteration of the training process, every K client conducts feedforward by the size of the mini-batch and send the output to the central server. The central server receives every output from the clients and propagates remaining layers with received outputs. Then the mini-batch size of central server will be

$$B = \sum_{k=1}^{K} b_k$$

As the mini-batch size of each client is proportional to its local data size, all client iterates the same number of times in each epoch. Additionally, if the dataset distribution for each client is unbalanced, our parallel method could leverage the effect of shuffling the dataset.

Client Layer Synchronization. By setting the mini-batch of each client proportional to its local data size, our method could process the training model in parallel. But in the training process gradient update of each client's layer is independent of each other. As a result, each client's weight becomes a different value at the end of the training. Also, these weights are all trained to fit their local data and cannot be used in the final model.

To solve these problems our method synchronize the layers of each client in initialize phase and in each iteration. The detailed workflow is presented in Algorithm 1. First, $client_1$ randomly initializes the weights of $L_{1,1}$. Every client synchronize the weight with $client_1$. Then every layer of clients $L_{k,1}$ has the same weights. After initialization, training process starts. Each client k propagates its layer $L_{k,1}$ with own predefined mini-batch and server continues propagation with clients' output. Server conducts backpropagation and every client back propagate with received error. As inputs of each client are different, gradient of each client layer are different. Every client transmits the gradients to server and server averages the received gradients. Averaged gradient are sent back to all clients and clients receive same gradient. As the layer weights of clients were same and gradients are same, updated weight of the clients will be same. By this process, clients synchronize the layer weights.

Algorithm 1: Parallel Split Learning with Client Synchronization

```
1:  Client_1 randomly initializes the weights of L_{1,1}.
2:  Every client synchronize the weight with Client_1.
3:  for epochs do
4:      for iterations do
5:          FeedForward in client layer L_{k,1}
6:          FeedForward in server layers L_2, ..., L_m
7:          BackPropagation in server
8:          BackPropagation in each client and get gradients
9:          Every client send gradient of L_{k,1} to server
10:         Server averages the received gradients
11:         Server sends averaged gradients to clients
12:         Clients update weights with received gradients
13:     end for
14: end for
```

III. Evaluation

To validate our approach, we measure the loss on each epoch and accuracy with our proposed method in split learning. For the baseline, we measure the loss and accuracy of the LeNet-5 model on MNIST data in one datacenter scenario. The same model and data were used to test with two scenarios in which the data size between clients is unbalanced. In the first scenario to test our approach, we distributed 60000 MNIST train data to 10 clients at rates of [0.6, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.02, 0.02, 0.01]. Then the dataset size for each client is [36000, 6000, 3000, 3000, 3000, 3000, 3000, 1200, 1200, 600] and mini-batch size is [60, 10, 5, 5, 5, 5, 5, 2, 2, 1] which is proportional to its dataset size. In this test scenario server mini-batch size B is 100. Every client iterates 600 times for each epoch. For the second scenario to test out approach, we distributed same data to 10 clients at rates of [0.9, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01], dataset size is [54000, 600, 600, 600, 600, 600, 600, 600, 600, 600] and mini-batch size is [90, 1, 1, 1, 1, 1 ,1 ,1 ,1, 1] which is proportional to its dataset size. Mini-batch size of server B in second scenario is 99. Every client iterates 600 times for each epoch. First and second scenario trained LeNet-5 model same as the baseline but first hidden layer belongs to the clients. Figure 2 shows reduction of loss in baseline and each scenario. Loss of our proposed method reduces similarly with the baseline.

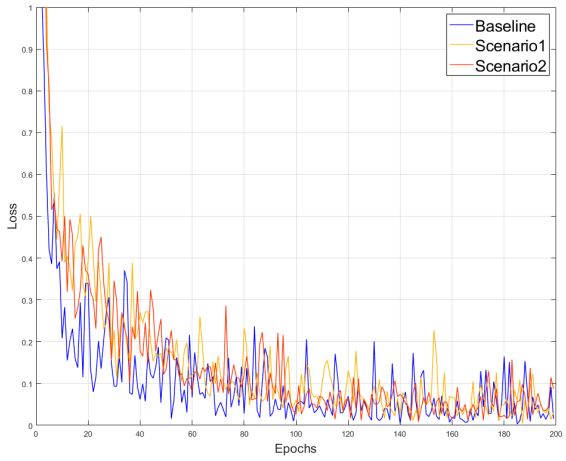

Fig. 2. Loss reduce for each epoch. Baseline and our proposed method's loss reduce similarly.

Scenario	Accuracy
Baseline	98.4
Scenario1	98.29
Scenario2	98.27

Table 1. Final accuracy of baseline and test scenarios

Table III shows the accuracy of trained deep learning model in baseline and each scenario. As shown in Table III, the

network converges to similar accuracies when training in distributed fashion.

IV. Conclusions and Future Work

We proposed a parallel method of split learning for deep learning model training with distributed datasets. We describe how to set the mini-batch of each client for parallel split learning. Then we provide client layer synchronization to conduct the final global model at the end of the training. We use popular dataset MNIST and LeNet-5 for validation of performance and show that out method produces similar result to the baseline. In future work, we could leverage split learning method to highly distributed environment like federated learning procedure. For that, split learning remains a problem to be solved, such as synchronization in a highly distributed environment.

Acknowledgment

This research was supported by the National Research Foundation of Korea (2019M3E4A1080391); and also by the Ministry of Health and Welfare (HI19C0572, HI19C0842). J. Kim is the corresponding author of this paper (joongheon@korea.ac.kr).

References

[1] J. Jeon, D. Kim, and J. Kim, "Cyclic parameter sharing for privacy-preserving distributed deep learning platforms," in *Proc. ICAIIC*, 2019.
[2] J. Chen, X. Pan, R. Monga, and S. Bengio, "Revisiting distributed synchronous SGD," in *arXiv preprint arXiv:1604.00981*, 2016.
[3] R. Shokri and V. Shmatikov, "Privacy-Preserving Deep Learning," in *Proc. ACM CCS*, October 2015.
[4] H.B. McMahan, E. Moore, D. Ramage, S. Hampson, and B.A.y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. AISTATS*, 2017.
[5] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, A. Dubey, "No Peek: A Survey of private distributed deep learning," *arXiv:1812.03288*, 2018.
[6] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv:1812.00564*, 2018.
[7] O. Gupta, R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, 2018.
[8] J. Jeon, J. Kim, J. Kim, K. Kim, A. Mohaisen and J. Kim, "Privacy-Preservind Deep Learning Computation for Geo-Distributed Medical Big-Data Platforms," in *IEEE/IFIP International Conference on Dependable Systems and Networks*, 2019.